# Formulario di Deep Learning

Mattia D'Urso

Università degli Studi di Udine — 26 gennaio 2021

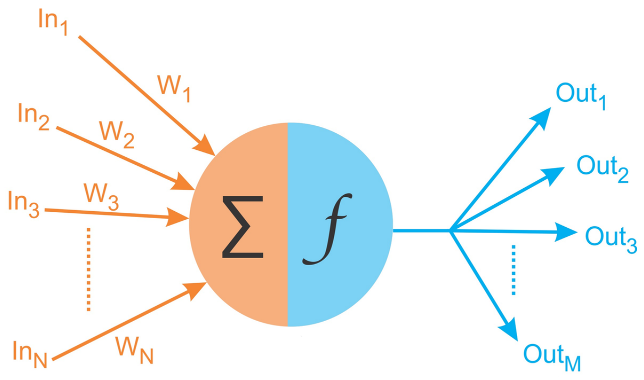
*Al prof. Serra ,*
*per avermi spinto incosapevolmente a scrivere questo formulario.*

# Indice dei contenuti

# 1 Introduzione

Questa raccolta di formule vuole essere un riassunto delle formule viste nel corso di Deep Learning 2020/2021 tenuto dal prof. Giuseppe Serra.

## 1.1 Formule base




$$h_{w,b}(X) = a = \sum W^T X + b$$

$$z = f(a)$$

con $f$ funzione di attivazione a piacere

## 1.2 Funzioni di attivazione

Sigmoid function [0,1]

$$Si(x) = \frac{1}{1+e^{-x}}$$

ReLU [0, x]

$$R(x) = max(0, x)$$

LeakyReLU [-x, x] con solitamente $\alpha = 0.2$

$$LR(x) = max(\alpha x, x)$$

Softmax [0,1]

$$So(x) = \frac{e^{z_i}}{\sum_{j=0} e^{z_j}}$$

Tanh [-1, 1]

$$Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

## 1.3 Funzioni di costo/loss

$y$ è la grand truth
$x$ sono gli input
$h_{w,b}(x_i)$ è la predizione

Mean Square Error (per la regression)

$$\mathcal{L}(w,b) = \frac{1}{m}\sum (h_{w,b}(x_i) - y_i)^2$$

Cross Entropy Loss (per la classificazione di più classi)

$$\mathcal{L}(w,b) = -\sum y_i ln(h_{w,b}(x_i))$$

Binary Cross Entropy Loss (per una classificazione binaria)

$$\mathcal{L}(w,b) = -\frac{1}{m}\sum y_i ln(h_{w,b}(x_i)) + (1-y_i)ln(1-h_{w,b}(x_i))$$

## 1.4 Gradient Descend

Con $\alpha$ il verso e $\beta$ l'intensità del vettore (oppure l'importanza che si da all'ultimo mini-batch usato, controlla quanto ogni nuovo mini-batch contribuisce alle medie correnti. Si può definire in più modi)

Gradient Descend vanilla

$$w_t = w_{t-1} - \alpha\frac{\partial\mathcal{L}}{\partial W}$$
$$b_t = b_{t-1} - \alpha\frac{\partial\mathcal{L}}{\partial b}$$

Gradient Descend with Momentum, $\beta = 0.9$

$$V_{dw} = \beta V_{t-1} + (1-\beta)\frac{\partial\mathcal{L}(w,b)}{\partial W}$$
$$V_{db} = \beta V_{t-1} + (1-\beta)\frac{\partial\mathcal{L}(w,b)}{\partial b}$$
$$w_t = w_{t-1} - \alpha V_{dw}$$
$$b_t = b_{t-1} - \alpha V_{db}$$

RSMprop, $\beta = 0.9$ e $\epsilon = 10^{-8}$

$$S_{dw} = \beta S_{dw} + (1-\beta)(\frac{\partial\mathcal{L}(w,b)}{\partial W})^2$$

$$S_{db} = \beta S_{db} + (1-\beta)(\frac{\partial\mathcal{L}(w,b)}{\partial b})^2$$

$$w_t = w_{t-1} - \alpha(\frac{\partial\mathcal{L}}{\partial W})/\sqrt{(\sigma^2+\epsilon)}$$

$$b_t = b_{t-1} - \alpha(\frac{\partial \mathcal{L}}{\partial b})/\sqrt{(\sigma^2 + \epsilon)}$$

ADAM $\beta_1 = 0.9$, $\beta_2 = 0.99$ e $\epsilon = 10^{-8}$

$$V_{dw} = \beta V_{t-1} + (1-\beta)\frac{\partial \mathcal{L}(w,b)}{\partial W}$$

$$V_{db} = \beta V_{t-1} + (1-\beta)\frac{\partial \mathcal{L}(w,b)}{\partial b}$$

$$S_{dw} = \beta S_{dw} + (1-\beta)(\frac{\partial \mathcal{L}(w,b)}{\partial W})^2$$

$$S_{db} = \beta S_{db} + (1-\beta)(\frac{\partial \mathcal{L}(w,b)}{\partial b})^2$$

$$w_t = w_{t-1} - \alpha(V_{dw})/\sqrt{(S_{dw} + \epsilon)}$$
$$b_t = b_{t-1} - \alpha(V_{db})/\sqrt{(S_{db} + \epsilon)}$$

## 1.5 Normalizzazione

### 1.5.1 Prima di usare il modello

minMAX normalization, da usare prima a meno che non si abbia una ragione teorica per aver bisogno di una normalizzazione più forte. [1]

$$x_{norm} = \frac{x - min(x)}{max(x) - min(x)}$$

Standard normalization, da usare quando è necessario trasformare una feature in modo che sia vicina alla distribuzione normale. [1]

$$x_{norm} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

con
$\mu$ la media
$\sigma$ la standard deviation al quadrato
$\epsilon$ un valore molto piccolo a piacere per evitare la divisone per 0
Dopo la normalizzazione $\mu = 1$ e $\sigma = 0$

Entrambe mantengono gli outliners, la prima ottiene valori in [0,1] la seconda ha un range più vario ma la distribuzione è centrata nell'origine
In generale vengono dati dei parametri $\gamma$ e $\beta$ tali che $x_{norm} = \gamma x + \beta$ che vengono poi aggiustati dalla backpropagation. Se la forma migliore per i dati è quella originaria convergono a $\gamma = \sqrt{\sigma^2 + \epsilon}$ e $\beta = \mu$.

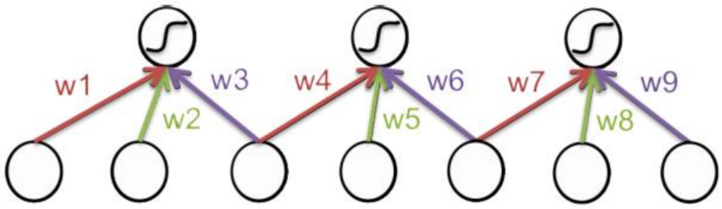
### 1.5.2 Dentro al modello

$$\mathcal{L}(w,b) = \frac{1}{m}[\sum_{i=1}^{m} Cost(h_{w,b}(x_i), y_i) + \frac{\lambda}{2}\sum_{j=1}^{n} w_j^2]$$

con
$\lambda$ che aumentando spinge i pesi verso zero.
Si usa questa tecnica per evitare che i pesi interni divergano troppo, ciò permette di usare lr più grandi e ridurre overfitting.

1. https://docs.google.com/spreadsheets/d/1woVi7wq13628HJ-tN6ApaRGVZ85OdmHsDBKLAf5ylaQ/edit#gid=0

## 1.6 Convolutional Neural Networks




Backpropagation in CNN, con $w_1 = w_4 = w_7$

$$w_1 = w_{t-1} - \alpha(\frac{\partial \mathcal{L}}{\partial W_1} + \frac{\partial \mathcal{L}}{\partial W_4} + \frac{\partial \mathcal{L}}{\partial W_7})$$

$$w_4 = w_{t-1} - \alpha(\frac{\partial \mathcal{L}}{\partial W_1} + \frac{\partial \mathcal{L}}{\partial W_4} + \frac{\partial \mathcal{L}}{\partial W_7})$$

$$w_7 = w_{t-1} - \alpha(\frac{\partial \mathcal{L}}{\partial W_1} + \frac{\partial \mathcal{L}}{\partial W_4} + \frac{\partial \mathcal{L}}{\partial W_7})$$

Calcolare numero di learning parameters nei livelli convolutivi

$$input * output + bias$$

input = # canali o filtri dell livello precedente
output = w del filtro * h del filtro
bias = # dei filtri

nella pratica

$$(w * h * previousFilters) + 1) * correntFilters$$

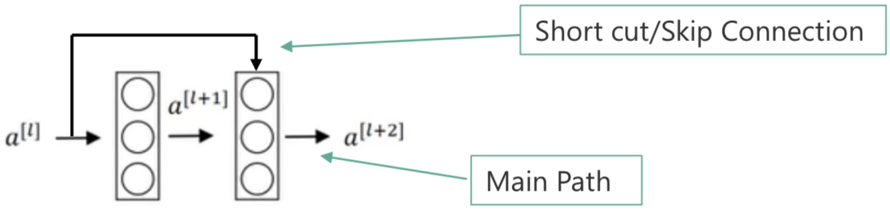
I layer pooling non hanno parametri

Fully Connected layer parameters

$$((previousLayerParameters * 1 + 1) * neurons)$$

Se si passa da un layer convolutivo o pool a uno fully connected si deve calcolare quanti parametri ha la matrice dell'immagine dopo le varie convoluzioni.

### 1.6.1 ResNet




$$a^{l+2} = g((W_2 a^{l+1} + b_2) + a^l)$$

## 1.7 Attention Models

Attention formula

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

con
$x$ = matrice dopo il positional encoding
$Q = x * W_Q$
$K = x * W_K$
$V = x * W_V$

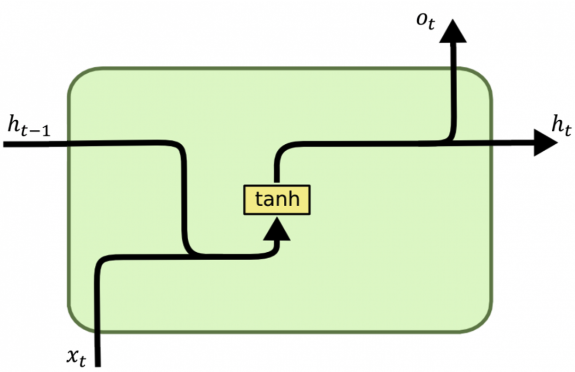
Il prodotto $QK^T$ viene fatto con la seguente formula

$$cos(v_i, u_j)||v_i||||u_j||$$

con
$v_i$ elemento di Q
$u_j$ elemento di K

## 1.8 Recurrent Neural Networks

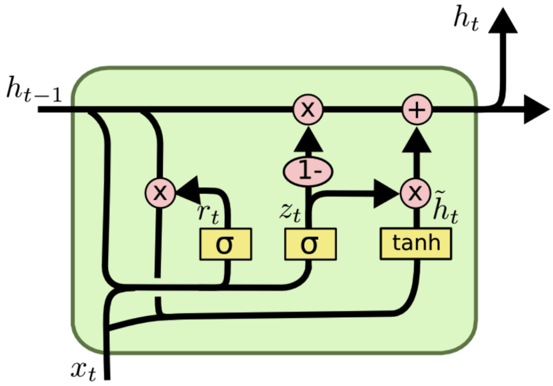
### 1.8.1 Vanilla RNN




$$a^{<t>} = tanh(W_a[a^{<t-1>}, x^t] + b_a)$$

con $a = h$ $\hat{y} = \sigma(W_y a^{<t>} + b_y)$

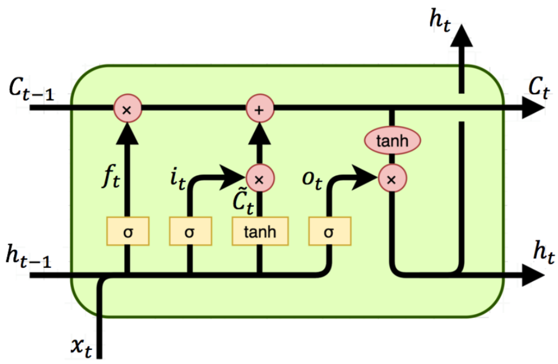
### 1.8.2 Full GRU RNN




$$\tilde{c}^{<t>} = tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$
$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$
$$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$$
$$c^{<t>} = \Gamma_u * \tilde{c}^{<}t > +(1 - \Gamma_u) * c^{<t-1>}$$
$$a^{<t>} = c^{<t>}$$

con $c = h$, $r_t = \Gamma_r$ e $z_t = \Gamma_u$

### 1.8.3 LSTM




$$\tilde{c}^{<t>} = tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$
$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$
$$\Gamma_f = \sigma(W_f[c^{<t-1>}, x^{<t>}] + b_f)$$
$$\Gamma_o = \sigma(W_o[c^{<t-1>}, x^{<t>}] + b_o)$$
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$
$$a^{<t>} = \Gamma_o * c^{<t>}$$

## 1.9 Generative Adversarial Networks

Discriminator Loss

$$minV(D) = E_{x \sim P_{data}(x)}[logD(x)] + E_{z \sim p_z(z)}[log(1 - D(G(z)))]$$

Generator Loss

$$E_{z \sim p_z(z)}[log(1 - D(G(z)))]$$

con
$z$ è il rumore di input
$G(z)$ il risultato del generatore
$D(G(z))$ il risultato del discriminatore
$E_x$ il valore atteso considerando tutti i dati reali(grand truth del discriminatore)
$E_z$ il valore atteso considerando tutti i dati reali del generatore(grand truth del generatore)

Ricordo che è un minMAX game e che se la accuracy dal generatore raggiunge il 100% il discriminatore arriva al massimo al 50% (con fake sample perfetti deve tirare a caso)

## 1.10 Graph Convolutional Networks

L'unica formula è

$$\hat{A} = (\tilde{D}^{\frac{1}{2}} * \tilde{A}) * \tilde{D}^{\frac{1}{2}}$$

e di conseguenza
$z = ReLU(W_0 * X_i' * \hat{A})$ nell'hidden layer e $z_n = Softmax(W_n * X_n' * \hat{A})$ nell'output layer

con
$\tilde{D} =$ la matrice dei gradi sommata ad $I$ alla $\frac{1}{2}$
$\tilde{A} =$ la matrice di adiacenza del grafo sommata a $\lambda I$